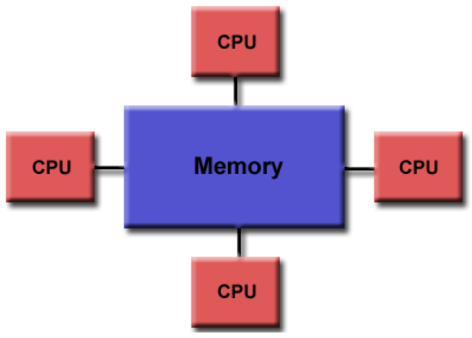
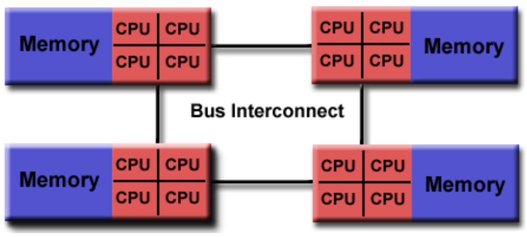
Parallel computing and OpenMP



Shared memory

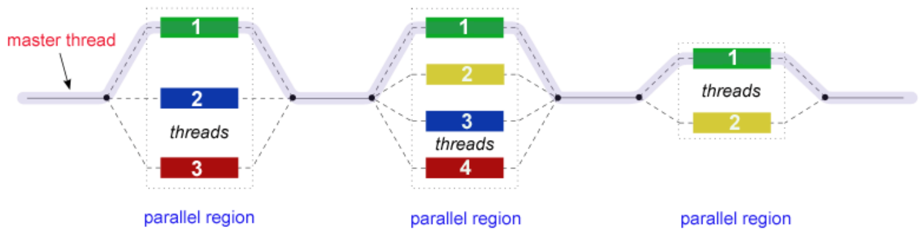


Distributed memory

So how does it work?

New comcept: `thread'

From a bundle of fibers, you can spawn a thread



Fork-join paradigm

That was theory, now practice

IMPORTANT: Large arrays should be dynamically allocated.

```
program arraytest
  use omp_lib
  implicit none
  integer, parameter :: N=50000000
  real(8),ALLOCATABLE :: vectors(:,:), norm_array(:)
  integer :: start_time, end_time
  integer(4) :: i

  ALLOCATE(Vectors(3,N),Norm_Array(N))
  call random_number(vectors) ! Fill array with random vectors
  call system_clock(start_time)
!$omp PARALLEL shared(vectors,norm_array) num_threads(2) &
!$omp private(i)
!$omp do schedule(static)
```

Spawn threads

start parallel do

NO ARRAY STATEMENTS!

```
  do i=1, n
    norm_array(i) = sum(vectors(:,i)*vectors(:,i)) ! Calculate
  end do
!$omp end do
```

end parallel do

```
!$omp do schedule(static)
```

New par. do

```
  do i=1, n
    vectors(:,i) = vectors(:,i)/sqrt(norm_array(i))
  end do
!$omp end do nowait
!$omp end parallel
```

Join master thread

```
  call system_clock(end_time)
  print '(I5)', end_time-start_time
!  vectors = vectors*spread(norm_array,1,3)
  print *, 'a', sum(vectors(:,500)*vectors(:,500),dim=1)
end program arraytest
```

Compiling and running your program

Compiler flag -fopenmp:

```
gfortran -O3 -march=native -ffast-math -fopenmp myprog.f90 -o myprog
```

Number of threads:

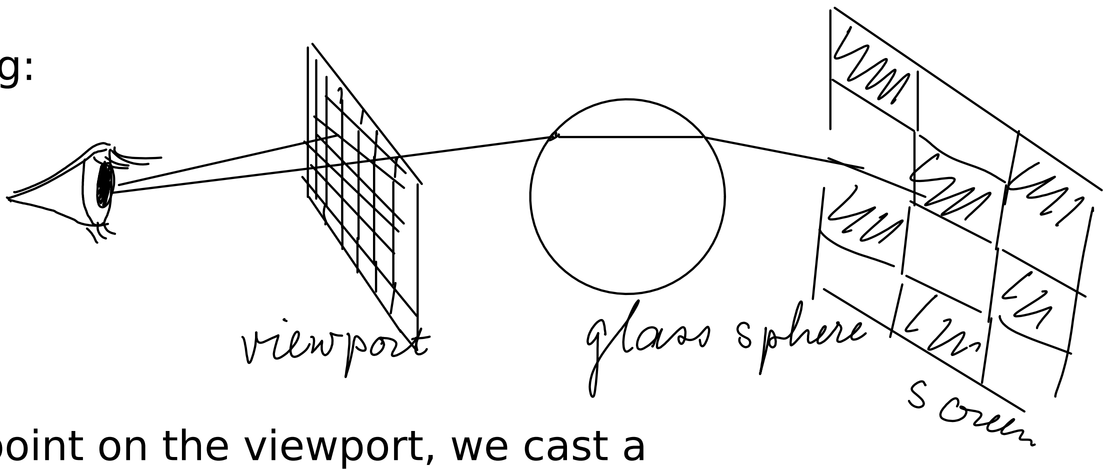
```
export OMP_NUM_THREADS=6  # specify that 6 threads will be used
```

The number of threads may exceed the number of processors!
This number can also be set inside the program (per fork)

Let's run the program end see how well it works!

Now for a `real' application

Ray-casting:



For each point on the viewport, we cast a ray and trace its path through the sphere using Snell's law

This is done independently for each pixel on the viewport:
parallel is easy!!!!

So what about private, shared?

Shared means: `global' variable
Private meand: `local' variable

Loop counters, help variables are usually private
Arrays to be manipulated are shared

In order to use OMP, you should always write out
array statements as loops!!!

What if you do not have loops, but different sections that may run in parallel?

https://computing.llnl.gov/tutorials/openMP/

You can use $OMP SECTIONS [shared,. private....]

Each section is then denoted by

```
$OMP SECTION
...
$OMP SECTION
....
....
```

SEE:

https://computing.llnl.gov/tutorials/openMP/